



Formation Doctrine

Romain Bohdanowicz

Twitter : @bioub

<http://formation.tech/>



Sommaire





Introduction



- ▶ **Romain Bohdanowicz**
Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle
- ▶ **Expérience**
Formateur/Développeur Freelance depuis 2006
Plus de 5000 heures de formation
- ▶ **Langages**
Expert : HTML / CSS / JavaScript / PHP / Java
Notions : C / C++ / Objective-C / C# / Python / Bash / Batch
- ▶ **Certifications**
PHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1
- ▶ **Et vous ?**
Langages ? Expérience ? Utilité de cette formation ?



► ODM ? ORM ? DBAL ?



Doctrine\Common



- Doctrine\Common

Ensemble des classes communes aux différents projets de Doctrine : DBAL, ORM, différents ODM...

- Regroupe en réalité plusieurs packages Composer :

- doctrine/lexer : contient la classe AbstractLexer permettant d'écrire un Lexer (utile pour définir une nouvelle syntaxe)
- doctrine/annotations : inclus doctrine/lexer et une bibliothèque pour créer et lire ses propres annotations
- doctrine/inflector : contient la classe Inflector qui permet de passer de table_name (snake_case) à tableName (camelCase) et inversement + conversions singuliers <-> pluriels
- doctrine/cache : contient les implémentations de Cache
- doctrine/collections : contient principalement la classe ArrayCollection pour manipuler/rechercher dans des tableaux
- doctrine/common : inclus tous les précédents + principalement Autoloader, EventManager, Reflection, Génération des Proxy



Doctrine\Common\Annotations



- ▶ Doctrine\Common\Annotations

Permet de manipuler des annotations personnalisés, celles qui seront utilisées plus tard dans les ORM et ODM.

- ▶ Installation

`composer require doctrine/annotations`

- ▶ Exemple

Création d'une bibliothèque permettant de générer un jeu de test et son annotation `FirstName` pour piocher aléatoire dans une liste de prénoms

```
<?php

namespace FormationTech\Entity;

use FormationTech\Annotation\FirstName;

class Contact
{
    /** @FirstName(language="en") */
    protected $prenom;

    // ... getter/setters ...
}
```



► Déclarer une annotation

Une annotation est une classe, qui est elle même annotée à minima avec l'annotation `@Annotation`

```
<?php

namespace FormationTech\Annotation;

/**
 * @Annotation
 */
class FirstName
{
}
```

```
<?php

namespace FormationTech\Entity;

use FormationTech\Annotation\FirstName;

class Contact
{
    /** @FirstName */
    protected $prenom;
}
```



▸ Annotation paramétrée

Pour paramétrer une annotation il suffit d'y déclarer des propriétés publiques

```
<?php

namespace FormationTech\Annotation;

/**
 * @Annotation
 */
class FirstName
{
    /** @var string */
    public $language = 'fr';
}
```

```
<?php

namespace FormationTech\Entity;

use FormationTech\Annotation\FirstName;

class Contact
{
    /** @FirstName(language="en") */
    protected $prenom;
}
```



- Annotation paramétrée avec un constructeur

```
/** @Annotation */
class LastName
{
    protected $languages = [];

    public function __construct(array $values)
    {
        // Example : @LastName(language="fr")
        if (isset($values['language']) && is_string($values['language'])) {
            $this->languages[] = $values['language'];
        }

        // Example : @LastName(languages={"fr", "en"})
        if (isset($values['languages']) && is_array($values['languages'])) {
            $this->languages += $values['languages'];
        }

        // Example : @LastName("fr")
        if (isset($values['value']) && is_string($values['value'])) {
            $this->languages[] = $values['value'];
        }

        // Example : @LastName({"fr", "en"})
        if (isset($values['value']) && is_array($values['value'])) {
            $this->languages += $values['value'];
        }
    }
}
```



► Cibler une annotation

Une annotation peut s'appliquer à une classe, une propriété, une méthode, etc... Pour cibler son utilisation on utilise l'annotation `@Target`, valeurs possibles : ALL, CLASS, METHOD, PROPERTY, ANNOTATION (pour les annotations imbriquées, cf `@JoinColumn`)

```
<?php

namespace FormationTech\Annotation;

/**
 * @Annotation
 * @Target("PROPERTY")
 */
class FirstName
{
}
```

```
<?php

namespace FormationTech\Annotation;

/**
 * @Annotation
 * @Target({"CLASS", "PROPERTY"})
 */
class Language
{
}
```



► Charger ses annotations

Les annotations Doctrine ne chargent pas automatiquement avec les autoloaders traditionnels (Composer, Zend, Symfony...), il faut soit les charger manuellement, soit utiliser :

`\Doctrine\Common\Annotations\AnnotationRegistry::registerAutoloadNamespace`
(PSR-0 uniquement)

```
<?php  
  
require_once 'vendor/autoload.php';  
  
use Doctrine\Common\Annotations\AnnotationRegistry;  
  
AnnotationRegistry::registerAutoloadNamespace(  
    'FormationTech\Annotation\\', __DIR__ . '/src'  
);
```



▸ Lire ses annotations

On utilise une classe Reader : AnnotationReader ou SimpleAnnotationReader (permet d'éviter de faire les use), qui peuvent être décorées par FileCacheReader, CachedReader ou IndexedReader pour la performance.

```
$reader = new \Doctrine\Common\Annotations\AnnotationReader();
```

```
$reader = new \Doctrine\Common\Annotations\SimpleAnnotationReader();  
$reader->addNamespace('FormationTech\Annotation');
```

```
$reader = new CachedReader(new AnnotationReader(), new ArrayCache())
```



```
<?php

namespace FormationTech\Driver;

use Doctrine\Common\Annotations\Reader;
use FormationTech\Annotation\FirstName;
use FormationTech\Provider\FirstNameProvider;

class AnnotationDriver
{
    /** @var Reader */
    protected $reader;

    public function __construct(Reader $reader)
    {
        $this->reader = $reader;
    }

    public function getFixture($className) {
        $class = new \ReflectionClass($className);

        $properties = $class->getProperties();

        $entity = new $className;

        foreach ($properties as $property) {
            $firstNameAnnotation = $this->reader->getPropertyAnnotation($property, FirstName::class);

            if ($firstNameAnnotation instanceof FirstName) {
                $provider = new FirstNameProvider();

                $setter = 'set' . $property->getName();

                if (!method_exists($entity, $setter)) {
                    throw new \Exception("Undefined setter $setter");
                }

                $entity->$setter($provider->getValue($firstNameAnnotation->language));
            }
        }

        return $entity;
    }
}
```


Doctrine\Common\Annotations



```
<?php

require_once 'vendor/autoload.php';

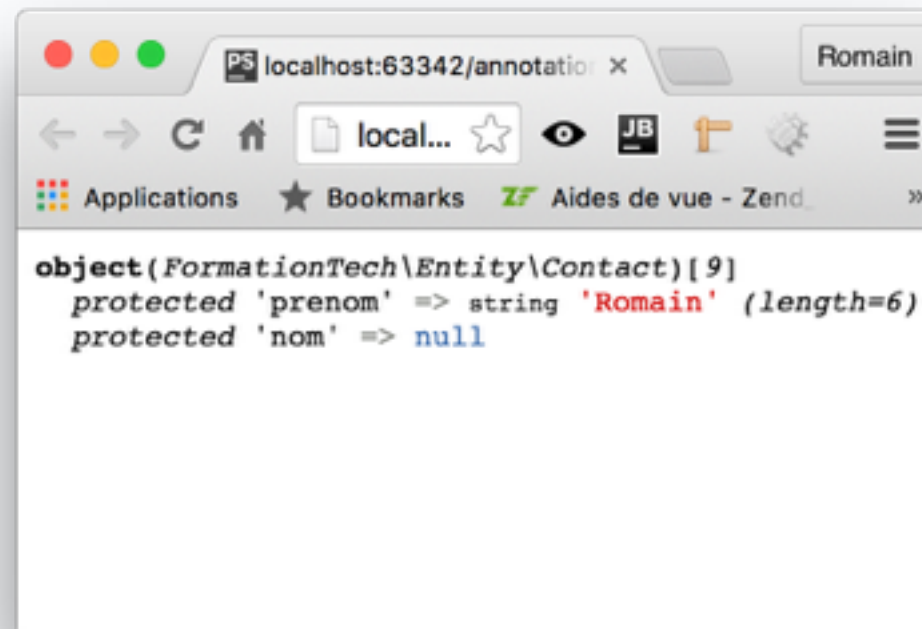
use Doctrine\Common\Annotations\AnnotationRegistry;

AnnotationRegistry::registerAutoloadNamespace(
    'FormationTech\Annotation\\', __DIR__ . '/src'
);

$reader = new \Doctrine\Common\Annotations\SimpleAnnotationReader();
$reader->addNamespace('FormationTech\Annotation');
$driver = new \FormationTech\Driver\AnnotationDriver($reader);

$contact = $driver->getFixture(\FormationTech\Entity>Contact::class);

var_dump($contact);
```





Doctrine\Common\Collections



- ▶ Doctrine\Common\Collections

Contient des classes pour manipuler et rechercher dans des collections d'objets.

- ▶ Installation

composer require doctrine/collections

```
<?php
```

```
require_once 'vendor/autoload.php';
```

```
$collection = new \Doctrine\Common\Collections\ArrayCollection();  
$collection->add('Romain');  
$collection->add('Eric');  
$collection->add('Jean');
```



▸ Boucler

```
foreach ($collection as $prenom) {  
    echo $prenom . '<br>';  
}  
  
$collection->forAll(function ($i, $prenom) {  
    echo $prenom . '<br>';  
    return true;  
});
```

▸ Contient

```
if ($collection->contains('Romain')) {  
    echo 'Contient Romain<br>';  
}
```

▸ Supprimer

```
$collection->remove(0);  
$collection->removeElement('Eric');  
$collection->clear();
```



► Filtrer

```
$collection->add(1);  
$collection->add(2);  
$collection->add(3);  
$collection->add(4);  
$collection->add(5);  
  
$pairs = $collection->filter(function ($nb) {  
    return $nb % 2 === 0;  
});  
  
var_dump($pairs->toArray());
```



► Rechercher des objets par critère

```
<?php
```

```
use Doctrine\Common\Collections\ArrayCollection;  
use Doctrine\Common\Collections\Criteria;  
use FormationTech\Entity>Contact;
```

```
require_once 'vendor/autoload.php';
```

```
$objectCollection = new ArrayCollection();  
$objectCollection->add((new Contact())->setPrenom('Romain')->setNom('Bohdanowicz')->setAge(30));  
$objectCollection->add((new Contact())->setPrenom('Barack')->setNom('Obama')->setAge(54));  
$objectCollection->add((new Contact())->setPrenom('Thierry')->setNom('Henry')->setAge(38));  
$objectCollection->add((new Contact())->setPrenom('Stephen')->setNom('Curry')->setAge(28));
```

```
$jeunes = $objectCollection->matching(new Criteria(Criteria::expr()->lt('age', 35)));
```

```
var_dump($jeunes->toArray());
```



Doctrine\Common\Cache



- ▶ Doctrine\Common\Cache
Contient différentes implémentations de cache clés/valeurs
- ▶ Installation
composer require doctrine/cache

```
<?php

require_once 'vendor/autoload.php';

$arrayCache = new \Doctrine\Common\Cache\ArrayCache();
$arrayCache->
```

m	contains(id : string)	bool
m	delete(id : string)	bool
m	deleteAll()	bool
m	fetch(id : string)	false mixed
m	fetchMultiple(keys : array)	array mixed[]
m	flushAll()	bool
m	getNamespace()	string
m	getStats()	array null
m	save(id : string, data : mixed, [lifeTime : int = 0])	bool
m	saveMultiple(keysAndValues : array, [lifetime : int = 0])	bool
m	setNamespace(namespace : string)	void

Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >> π



- Implémentations (version 1.6)
 - ApcCache (nécessite ext/apc)
 - ApcuCache (nécessite ext/apcu)
 - ArrayCache (en mémoire, durée de vie de la requête)
 - CouchbaseCache (nécessite ext/couchbase)
 - FilesystemCache (pas optimal en cas de forte concurrence)
 - MemcacheCache (nécessite ext/memcache)
 - MemcachedCache (nécessite ext/memcached)
 - MongoDBCache (nécessite ext/mongo)
 - PhpFileCache (pas optimal en cas de forte concurrence)
 - PRedisCache.php (nécessite predis/predis)
 - RedisCache.php (nécessite ext/phpredis)
 - RiakCache (nécessite ext/riak)
 - SQLite3Cache (nécessite ext/sqlite)
 - VoidCache (utile pour les tests)
 - WinCacheCache.php (nécessite ext/wincache)
 - XcacheCache.php (nécessite ext/xcache)
 - ZendDataCache.php (nécessite Zend Server Platform)



- ▶ **Norme PSR-6**

Depuis peu il existe une norme cherchant à uniformiser les systèmes de cache. Permettrait par exemple de rendre compatible zendframework/zend-cache avec doctrine/cache.

<http://www.php-fig.org/psr/psr-6/>

- ▶ **Doctrine Adapter**

Le projet PHP-Cache propose une compatibilité avec PSR-6 ce qui n'est pas le cache de doctrine + un adaptateur pour pouvoir utiliser les classes de doctrines sur une application PSR-6 et inversement.

<http://www.php-cache.com/en/latest/>



Doctrine\DBAL



▸ Doctrine\DBAL

Doctrine DBAL (database abstraction & access layer) est un API similaire à PDO qui lui ajoute des fonctionnalités comme l'introspection et la manipulation de bases et de tables.

▸ Installation

`composer require doctrine/dbal`

▸ SGBD supportés

- MySQL
- Oracle
- Microsoft SQL Server
- PostgreSQL
- SAP Sybase SQL Anywhere SQLite
- Drizzle



- 2 classes principales
 - Doctrine\DBAL\Connection
Equivalent à PDO
 - Doctrine\DBAL\Statement
Equivalent à PDOStatement



▸ Manipuler des bases de données

```
<?php

require_once 'vendor/autoload.php';

$connectionParams = [
    'user' => 'root',
    'password' => '',
    'host' => 'localhost',
    'driver' => 'pdo_mysql',
];

$conn = \Doctrine\DBAL\DriverManager::getConnection($connectionParams);

$sm = $conn->getSchemaManager();
$sm->createDatabase('tests_doctrine');

var_dump($sm->listDatabases());
```



► Créer des tables

```
$table = new \Doctrine\DBAL\Schema\Table('contact');
$table->addColumn('id', 'integer', ['autoincrement' => true]);
$table->addColumn('prenom', 'string', ['length' => 40]);
$table->addColumn('nom', 'string', ['length' => 40]);
$table->addColumn('email', 'string', ['length' => 80, 'notNull' => false]);
$table->addColumn('telephone', 'string', ['length' => 20, 'notNull' => false]);
$table->addColumn('date_naissance', 'date', ['notNull' => false]);
$table->setPrimaryKey(['id']);

// Pour afficher la requête
var_dump($sm->getDatabasePlatform()->getCreateTableSQL($table));

// Pour créer la table
$sm->createTable($table);
```



▸ Recherches

- Avec un API style PDO

```
$stmt = $conn->executeQuery('SELECT * FROM contact');  
var_dump($stmt->fetchAll(PDO::FETCH_ASSOC));
```

- Avec un Query Builder

```
$qb = $conn->createQueryBuilder();  
  
$qb->select('prenom', 'nom')  
    ->from('contact')  
    ->orderBy('prenom', 'ASC');  
  
$stmt = $qb->execute();  
var_dump($stmt->fetchAll(PDO::FETCH_ASSOC));
```




► Différences entre 2 bases de données (scripts de migration)

```
$oldTable = new \Doctrine\DBAL\Schema\Table('contact');
$oldTable->addColumn('prenom', 'string');
$oldTable->addColumn('nom', 'string');
$oldSchema = new \Doctrine\DBAL\Schema\Schema([$oldTable]);

$newTable = new \Doctrine\DBAL\Schema\Table('contact');
$newTable->addColumn('prenom', 'string');
$newTable->addColumn('nom', 'string');
$newTable->addColumn('email', 'string', ['notNull' => false]);
$newTable->addColumn('telephone', 'string', ['notNull' => false]);
$newSchema = new \Doctrine\DBAL\Schema\Schema([$newTable]);

$dbPlatform = $conn->getDatabasePlatform();

$comparator = new \Doctrine\DBAL\Schema\Comparator();
$diff = $comparator->compare($oldSchema, $newSchema);
var_dump($diff->toSql($dbPlatform));

$comparator = new \Doctrine\DBAL\Schema\Comparator();
$diff = $comparator->compare($newSchema, $oldSchema);
var_dump($diff->toSql($dbPlatform));
```



Doctrine\ORM



▶ Doctrine\ORM

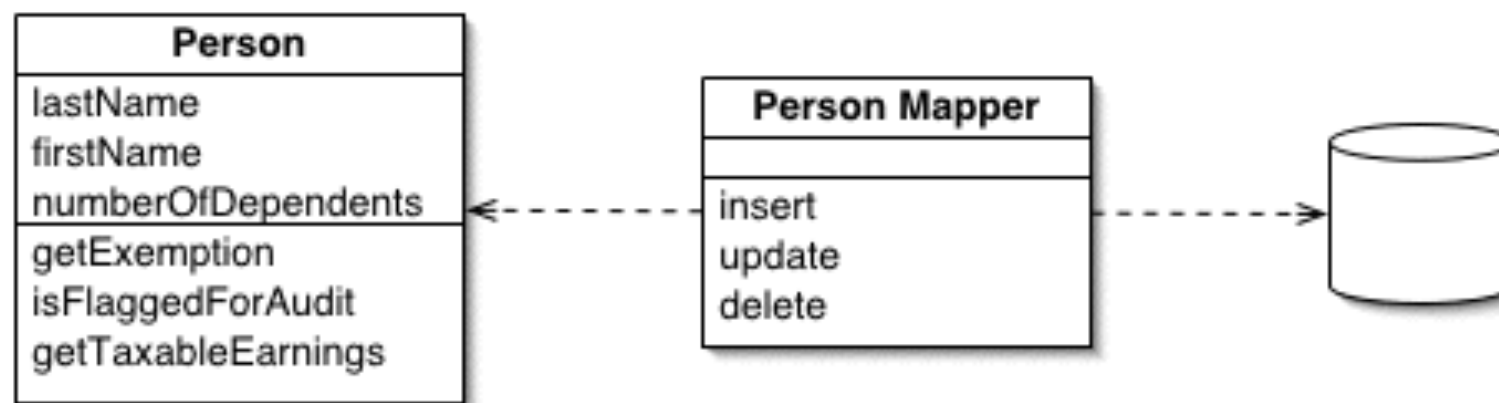
Doctrine ORM (object relational mapping) est un API permettant de faire traduire des manipulations d'objets en requêtes SQL.

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/>

▶ Data Mapper

Doctrine ORM est une implémentation du Design Pattern Data Mapper, c'est un composant qui permet de communiquer avec une base de données de manière objet. Il est inspiré de la bibliothèque Hibernate en Java.

<http://martinfowler.com/eaCatalog/dataMapper.html>



▶ Installation

`composer require doctrine/orm`



- ▶ **Doctrine\ORM\EntityManager**
L'objet responsable de la persistance des entités.
- ▶ **Doctrine\ORM\EntityRepository**
L'objet responsable de la récupération des entités.
- ▶ **Portabilité vers Doctrine\ODM**
Pour être portable vers Doctrine\ODM, utiliser les interfaces Doctrine\Common\Persistence\ObjectManager et Doctrine\Common\Persistence\ObjectRepository dans les DocBlocks.



▸ Configuration du Mapping

Le mapping est la traduction entre le monde objet et le monde de la base de données.

▸ Formats possibles

- Annotations
- XML
- YAML



```
<?php

namespace AddressBook\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Contact
 *
 * @ORM\Table(name="contact", uniqueConstraints={@ORM\UniqueConstraint(name="email_UNIQUE", columns={"email"})},
indexes={@ORM\Index(name="fk_contact_societe_idx", columns={"societe_id"}), @ORM\Index(name="fk_contact_membre1_idx",
columns={"membre_id"})})
 * @ORM\Entity
 */
class Contact
{
}
}
```

- ▶ **@ORM\Entity**
Pour déclarer la classe persistente
- ▶ **@ORM\Table(name="contact")**
Pour déclarer à quelle table est associée cette entité
Permet également de définir des index et contrainte unique.



▸ @ORM\Column

Pour lier une propriété à une colonne, attributs **name** id le nom de colonne est différent, **length** taille, **nullable** (true/false), **type** parmi :

- **string** (string <> VARCHAR)
- **integer** (int <> INT)
- **smallint** (int <> SMALLINT)
- **bigint** (string <> BIGINT)
- **boolean** (boolean)
- **decimal** (float <> DECIMAL avec des options precision et scale)
- **date** (\DateTime <> DATETIME)
- **time** (\DateTime <> TIME)
- **datetime** (\DateTime <> DATETIME/TIMESTAMP)
- **text** (string <> TEXT), object (serialize(object) <> unserialize(TEXT))
- **array** (serialize(object) <> unserialize(TEXT))
- **float** (float <> FLOAT (séparateur décimale .))

▸ Il est également possible de définir ses propres types (peut-être utiliser pour les ENUM) :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/cookbook/mysql-enums.html>



- ▶ `@ORM\Id`

Si la colonne est la primaire.

- ▶ `@ORM\GeneratedValue(strategy="IDENTITY")`

Si la clé primaire est générée par le SGBDR (`strategy="IDENTITY"` pour MySQL/SQLite/MSSQL), (`strategy="SEQUENCE"` pour PostgreSQL/Oracle), (`strategy="AUTO"` pour être portable)



► Exemple :

```
/**
 * @var integer
 *
 * @ORM\Column(name="id", type="integer", nullable=false)
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="IDENTITY")
 */
private $id;

/**
 * @var string
 *
 * @ORM\Column(name="prenom", type="string", length=45, nullable=false)
 */
private $prenom;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="date_naissance", type="date", nullable=true)
 */
private $dateNaissance;

/**
 * @var int
 *
 * @ORM\Column(name="taille", type="integer", nullable=true)
 */
private $taille;
```



▸ Associations

Doctrine ORM permet de définir directement les associations entre les entités et s'occupera de faire la plupart des jointures et insertions multiples automatiquement.

- **OneToOne**

Relation 1..1

- **OneToMany**

Relation 1..n du côté 1

- **ManyToOne**

Relation 1..n du côté n (du côté de la clé étrangère)

- **ManyToMany**

Relation n..m

▸ Unidirectionnelle / Bidirectionnelle

La relation peut-être unidirectionnelle (une entité en connaît une autre mais l'inverse n'est pas vrai) ou bidirectionnelle (chaque entité connaît l'autre).

Les relations bidirectionnelles doivent déclarer la relation opposées (mappedBy/inversedBy).



```
class Contact
{
    /**
     * @var \Application\Entity\Membre
     *
     * @ORM\OneToOne(targetEntity="AddressBook\Entity\Membre")
     * @ORM\JoinColumns({
     *     @ORM\JoinColumn(name="membre_id", referencedColumnName="id")
     * })
     */
    private $membre;

    /**
     * @var \Application\Entity\Societe
     *
     * @ORM\ManyToOne(targetEntity="Application\Entity\Societe", inversedBy="contacts")
     * @ORM\JoinColumns({
     *     @ORM\JoinColumn(name="societe_id", referencedColumnName="id")
     * })
     */
    private $societe;

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\ManyToMany(targetEntity="Application\Entity\Association")
     * @ORM\JoinTable(name="adhesion",
     *     joinColumns={
     *         @ORM\JoinColumn(name="contact_id", referencedColumnName="id")
     *     },
     *     inverseJoinColumns={
     *         @ORM\JoinColumn(name="association_id", referencedColumnName="id")
     *     }
     * )
     */
    private $associations;
}
```



▶ Exemple AddressBook\Entity\Societe :

```
class Societe
{
    // ...

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\OneToMany(targetEntity="Application\Entity>Contact", mappedBy="societe")
     */
    private $contacts;

    /**
     * Constructor
     */
    public function __construct()
    {
        $this->contacts = new \Doctrine\Common\Collections\ArrayCollection();
    }
}
```



► bootstrap.php

Création de l'EntityManager (\$isDevMode pour désactiver le cache)

```
<?php
// bootstrap.php
require_once "vendor/autoload.php";

// Create a simple "default" Doctrine ORM configuration for XML Mapping
$isDevMode = true;
$config = \Doctrine\ORM\Tools
\Setup::createAnnotationMetadataConfiguration(array(__DIR__."/src"), $isDevMode);
// or if you prefer yaml or annotations
// $config = Setup::createXMLMetadataConfiguration(array(__DIR__."/config/xml"),
// $isDevMode);
// $config = Setup::createYAMLMetadataConfiguration(array(__DIR__."/config/yaml"),
// $isDevMode);

// database configuration parameters
$conn = array(
    'driver' => 'pdo_sqlite',
    'path' => __DIR__ . '/db.sqlite',
);

// obtaining the entity manager
$entityManager = \Doctrine\ORM\EntityManager::create($conn, $config);
```



► cli-config.php

Pour pouvoir utiliser les binaires de Doctrine\ORM il faut créer un fichier cli-config à la racine du projet ou dans un répertoire config.

```
<?php
// cli-config.php
require_once "bootstrap.php";

$helperSet = new \Symfony\Component\Console\Helper\HelperSet(array(
    'em' => new \Doctrine\ORM\Tools\Console\Helper\EntityManagerHelper($entityManager),
    'db' => new \Doctrine\DBAL\Tools\Console\Helper\ConnectionHelper(\Doctrine\DBAL
\DriverManager::getConnection($conn))
));

return $helperSet;
```



► Insertions

```
<?php
// create_user.php
require_once "bootstrap.php";

$user = new User();
$user->setName('Romain');

$entityManager->persist($user);
$entityManager->flush();

echo "Created User with ID " . $user->getId() . "\n";
```



► Recherches

```
<?php
// list_bugs.php
require_once "bootstrap.php";

$dql = "SELECT b, e, r FROM Bug b JOIN b.engineer e JOIN b.reporter r ORDER BY b.created
DESC";

$query = $entityManager->createQuery($dql);
$query->setMaxResults(30);
$bugs = $query->getResult();

foreach($bugs AS $bug) {
    echo $bug->getDescription()." - ".$bug->getCreated()->format('d.m.Y')."\\n";
    echo "    Reported by: ".$bug->getReporter()->getName()."\\n";
    echo "    Assigned to: ".$bug->getEngineer()->getName()."\\n";
    foreach($bug->getProducts() AS $product) {
        echo "        Platform: ".$product->getName()."\\n";
    }
    echo "\\n";
}
```




► extends Doctrine\ORM\EntityRepository

Parfois les requêtes ne peuvent pas s'écrire où ne sont pas bien optimisés, il faut alors créer une classe Repository qui héritera de Doctrine\ORM\EntityRepository
Les requêtes peuvent alors s'écrire :

- En SQL

Il faudra alors expliquer à Doctrine comment cette requête se traduit en entité

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/native-sql.html>

- Avec le QueryBuilder

Equivalent de Zend\Db\Sql

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html>

- En DQL

Language propre à Doctrine proche de SQL mais manipulant des entités.

► Création du Repository

Modifier l'annotation @ORM\Entity d'une entité pour :

```
@ORM\Entity(repositoryClass="AddressBook\Entity\Repository\ContactRepository")
```

Exécuter la commande :

```
vendor/bin/doctrine-module orm:generate:repositories module/AddressBook/src
```

Doctrine - Requêtes « complexes »



► Exemple

Dans `showAction` du contrôleur `Contact`, nous requérons une entité par sa clé primaire. Dans la vue nous demandons d'accéder à sa Société liée.

Par défaut Doctrine fait 2 requêtes SQL, nous aimerions en faire 1 seule requêtes avec une jointure.

```
// AddressBook/Controller/ContactController
public function showAction()
{
    $id = $this->params("id");

    $contact = $this->getRepository()->find($id);

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

```
<!-- view/address-book/contact/show.phtml -->
<?=$this->escapeHtml($contact->getSociete()->getNom())?>
```

» DoctrineORMModule

» Queries for doctrine.sql_logger_collector.orm_default

SQL:

```
SELECT t0.id AS id1, t0.prenom AS prenom2, t0.nom AS nom3, t0.telephone AS
telephone4, t0.email AS email5, t0.membre_id AS membre_id6, t0.societe_id AS
societe_id7 FROM contact t0 WHERE t0.id = ?
```

Params: 0 => string '1' (length=1)

Types: 0 => string 'integer' (length=7)

Time: 0.00042605400085449

SQL:

```
SELECT t0.id AS id1, t0.nom AS nom2, t0.ville AS ville3 FROM societe t0
WHERE t0.id = ?
```

Params: 0 => int 3



2 queries in 876.19 µs



4 mappings

Doctrine - Requêtes « complexes »



► Exemple

Avec le Repository, on obtient une seule requête (temps d'exécution divisé par 2).

```
// AddressBook/Entity/Repository/ContactRepository

class ContactRepository extends EntityRepository
{
    public function findWithSociete($id)
    {
        $dql = "SELECT c, s
                FROM AddressBook\Entity\Contact c
                LEFT JOIN c.societe s
                WHERE c.id = :id";

        return $this->getEntityManager()->createQuery($dql)
            ->setParameter("id", $id)
            ->getSingleResult();
    }
}
```

```
// AddressBook/Controller/ContactController

public function showAction()
{
    $id = $this->params("id");

    $contact = $this->getRepository()->findWithSociete($id);

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

```
» DoctrineORMModule
» Queries for doctrine.sql_logger_collector.orm_default
```

```
SQL:
SELECT c0_.id AS id0, c0_.prenom AS prenom1, c0_.nom AS nom2,
c0_.telephone AS telephone3, c0_.email AS email4, s1_.id AS id5, s1_.nom AS
nom6, s1_.ville AS ville7, c0_.membre_id AS membre_id8, c0_.societe_id AS
societe_id9 FROM contact c0_ LEFT JOIN societe s1_ ON c0_.societe_id =
s1_.id WHERE c0_.id = ?

Params:          0 => string '1' (length=1)
Types:           0 => int 2
Time:            0.0004878044128418
```

1 queries in 487.80 µs 4 mappings



DoctrineModule et DoctrineORMModule



- ▶ DoctrineModule et DoctrineORMModule

Modules d'intégration pour Zend Framework 2 et 3. DoctrineModule contient les composants communs à l'ORM et l'ODM, DoctrineORMModule les spécificités de l'ORM.

<https://github.com/doctrine/DoctrineModule>

<https://github.com/doctrine/DoctrineORMModule>

- ▶ Installation

`composer require doctrine/doctrine-orm-module`

- ▶ Activation du module

```
<?php

$modules = array(
    'DoctrineModule',
    'DoctrineORMModule',
    'Application',
);

return array(
    'modules' => $modules,

    // ...
```



► Configuration

Mettre le spécifique au module dans la config du module et le global dans config/autoload.

```
<?php
// config/autoload/doctrine.global.php
return [
    'doctrine' => [
        'connection' => [
            'orm_default' => [
                'driverClass' => Doctrine\DBAL\Driver\PDOMySql\Driver::class,
                'params' => [
                    'host' => 'localhost',
                    'port' => '3306',
                    'user' => 'username',
                    'password' => 'password',
                    'dbname' => 'database',
                ]
            ]
        ]
    ],
    'driver' => [
        'my_annotation_driver' => [
            'class' => 'Doctrine\ORM\Mapping\Driver\AnnotationDriver',
            'cache' => 'array',
        ]
    ]
];
```



► Configuration

```
<?php
// module/Application/config/doctrine.config.php
return [
    'doctrine' => [
        'driver' => [
            'my_annotation_driver' => [
                'paths' => [
                    __DIR__ . '/../src/Application/Entity'
                ],
            ],
        ],
        'orm_default' => [
            'drivers' => [
                'Application\Entity' => 'my_annotation_driver'
            ]
        ]
    ]
],
];
```

DoctrineModule et DoctrineORMModule



- Ligne de commande
vendor/bin/doctrine-module

DoctrineModule Command Line Interface version 1.0.1

Usage:

command [options] [arguments]

Options:

-h, --help	Display this help message
-q, --quiet	Do not output any message
-V, --version	Display this application version
--ansi	Force ANSI output
--no-ansi	Disable ANSI output
-n, --no-interaction	Do not ask any interactive question
-v vv vvv, --verbose	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands:

help	Displays help for a command
list	Lists commands
dbal	
dbal:import	Import SQL file(s) directly to Database.
dbal:run-sql	Executes arbitrary SQL directly from the command line.

DoctrineModule et DoctrineORMModule



orm

<code>orm:clear-cache:metadata</code>	Clear all metadata cache of the various cache drivers.
<code>orm:clear-cache:query</code>	Clear all query cache of the various cache drivers.
<code>orm:clear-cache:result</code>	Clear all result cache of the various cache drivers.
<code>orm:convert-d1-schema</code>	Converts Doctrine 1.X schema into a Doctrine 2.X schema.
<code>orm:convert-mapping</code>	Convert mapping information between supported formats.
<code>orm:convert:d1-schema</code>	Converts Doctrine 1.X schema into a Doctrine 2.X schema.
<code>orm:convert:mapping</code>	Convert mapping information between supported formats.
<code>orm:ensure-production-settings</code>	Verify that Doctrine is properly configured for a production environment.
<code>orm:generate-entities</code>	Generate entity classes and method stubs from your mapping information.
<code>orm:generate-proxies</code>	Generates proxy classes for entity classes.
<code>orm:generate-repositories</code>	Generate repository classes from your mapping information.
<code>orm:generate:entities</code>	Generate entity classes and method stubs from your mapping information.
<code>orm:generate:proxies</code>	Generates proxy classes for entity classes.
<code>orm:generate:repositories</code>	Generate repository classes from your mapping information.
<code>orm:info</code>	Show basic information about all mapped entities
<code>orm:run-dql</code>	Executes arbitrary DQL directly from the command line.
<code>orm:schema-tool:create</code>	Processes the schema and either create it directly on EntityManager Storage Connection or generate the SQL output.
<code>orm:schema-tool:drop</code>	Drop the complete database schema of EntityManager Storage Connection or generate the corresponding SQL output.
<code>orm:schema-tool:update</code>	Executes (or dumps) the SQL needed to update the database schema to match the current mapping metadata.
<code>orm:validate-schema</code>	Validate the mapping files.



► Générer le mapping depuis une base de données

Le mapping peut être au format XML, YAML, Annotations ou PHP

```
./vendor/bin/doctrine-module orm:convert-mapping --from-database --  
namespace=Application\Entity\ annotation module/Application/src
```

- ```
./vendor/bin/doctrine-module orm:convert-mapping --from-database --filter="Contact|
Societe" --namespace=Application\Entity\ annotation module/Application/src
```

## ► Générer les entités depuis le mapping

Si le mapping est au format annotation alors Doctrine est obligé de créer les entités et leurs propriétés, cependant il manque constructeur et accesseurs.

```
vendor/bin/doctrine-module orm:generate-entities module/Application/src
```

## ► Générer la base de données depuis le mapping

Pour vérifier que la requête SQL est correcte

```
vendor/bin/doctrine-module orm:schema-tool:update --dump-sql
```

Pour exécuter la requête SQL

```
vendor/bin/doctrine-module orm:schema-tool:update --force
```



## ► Clés du ServiceManager

```
class ContactController extends ActionController
{
 // ...

 /**
 * @return \Doctrine\Common\Persistence\ObjectManager
 */
 public function getEntityManager()
 {
 return $this->getServiceLocator()->get('Doctrine\ORM\EntityManager');
 }

 /**
 * @return \Doctrine\Common\Persistence\ObjectRepository
 */
 public function getRepository()
 {
 return $this->getEntityManager()->getRepository('AddressBook\Entity>Contact');
 }
}
```